目录

[一、手写识别算法见习2018.04.09始 2](#_Toc27583)

[1.1：K-NN算法 错误率：0.032770 2](#_Toc10127)

[1.1.1：KNN分类算法思想 2](#_Toc23857)

[1.1.2：归一化数值算法思想及方法 3](#_Toc8906)

[1.1.3：数据准备 4](#_Toc27884)

[1.1.4：测试算法 4](#_Toc4718)

[1.1.5：运行结果 5](#_Toc22562)

[2.1：K-NN算法之TensorFlow框架之 线性处理 准确率: 0.91 6](#_Toc24205)

[2.1.1：TensorFlow手写数字识别大致步骤 6](#_Toc5675)

[2.1.2：数据集准备 6](#_Toc23139)

[2.1.3：输入和输出、损失函数 7](#_Toc10612)

[2.1.4：回归模型 7](#_Toc4678)

[2.1.5：递归下降、学习率 7](#_Toc6703)

[2.1.6：代码实现 8](#_Toc11788)

[2.1.7：运行结果 10](#_Toc22424)

[3.1：K-NN算法之TensorFlow框架之 CNN 处理 准确率: 1 11](#_Toc19218)

# 一、手写识别算法见习2018.04.09始

## 1.1：K-NN算法 错误率：**0.032770**

### 1.1.1：KNN分类算法思想

对未知类别属性的数据集中的每个点依次执行以下操作：

1. **计算已知类别数据集中的点与当前点之间的距离**
2. **按照距离升序排列**
3. **选取与当前点距离最小的K个点**
4. **确定前K个点所在类别的出现频率**
5. **返回前K个点出现频率最高的类别当做当前点的预测分类**

def **classify0**(inX, dataSet, labels, k):

dataSetSize = dataSet.shape[0]

diffMat = **tile(inX, (dataSetSize,1))** - dataSet

sqDiffMat = diffMat\*\*2

sqDistances = sqDiffMat.sum(axis=1)

distances = sqDistances\*\*0.5

sortedDistIndicies = distances.argsort()

classCount={}

for i in range(k):

voteIlabel = labels[sortedDistIndicies[i]]

classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1

sortedClassCount = sorted(classCount)

return sortedClassCount[0]

### [1.1.2：归一化数值算法思想及方法](https://www.cnblogs.com/sddai/p/6250094.html)

一般做机器学习应用的时候大部分时间是花费在特征处理上，其中很关键的一步就是对特征数据进行归一化，为什么要归一化呢？很多同学并未搞清楚，维基百科给出的解释：

1. 归一化后加快了**梯度下降求最优解的速度**;
2. 归一化**有可能提高精度**;

**归一化方法：**

1. **简单缩放 | min-max标准化(Min-max normalization) | 离差标准化**

在简单缩放中，我们的目的是通过对数据的每一个维度的值进行重新调节（这些维度可能是相互独立的），使得最终的数据向量落在 [0,1]或[ − 1,1] 的区间内（根据数据情况而定）。这对后续的处理十分重要，因为很多默认参数（如 PCA-白化中的 epsilon）都假定数据已被缩放到合理区间。 例子:在处理自然图像时，我们获得的像素值在 [0,255] 区间中，常用的处理是将这些像素值除以 255，使它们缩放到 [0,1] 中.这种算法是对原始数据的线性变换，使结果落到[0,1]区间，转换函数如下：

**x ＝ (x - min)/(max - min)**

max: 样本数据的最大值

min: 为样本数据的最小值

适用场景

这种归一化方法比较适用在数值比较集中的情况。但是，如果max和min不稳定，很容易使得归一化结果不稳定，使得后续使用效果也不稳定，实际使用中可以用经验常量值来替代max和min。而且当有新数据加入时，可能导致max和min的变化，需要重新定义。

在不涉及距离度量、协方差计算、数据不符合正太分布的时候，可以使用第一种方法或其他归一化方法。比如图像处理中，将RGB图像转换为灰度图像后将其值限定在[0 255]的范围

1. **标准差标准化 | z-score 0均值标准化(zero-mean normalization)**

经过处理的数据符合标准正态分布，即均值为0，标准差为1，其转化函数为：

x = (x - u)/σ

* u: 所有样本数据的均值
* σ: 为所有样本数据的标准差。

在分类、聚类算法中，需要使用距离来度量相似性的时候、或者使用PCA技术进行降维的时候，第二种方法(Z-score standardization)表现更好。

def **autoNorm**(dataSet):

# dataSet数据集中每列中的最小值，参数0代表是列数

minvals=dataSet.min(0)

maxvals=dataSet.max(0)

ranges=maxvals-minvals

normDataSet=zeros(np.shape(dataSet))

m=dataSet.shape[0]

*'''tile()函数与将变量内容复制成输入矩阵同样大小的矩阵'''*

normDataSet=dataSet-tile(minvals,(m,1))

normDataSet=normDataSet/tile(ranges, (m,1))

return normDataSet,ranges,minvals

### 1.1.3：数据准备

目录在trainingDigits包含了2000个例子如图所示：每个数字大约有200个，测试集有900个。

首先，将图像格式化处理为一个向量，把32X32转换成1X1024的向量

def **img2vector**(filename):

returnVect=zeros((1,1024))

fr=open(filename)

for i in range(32):

linestr=fr.readline()

for j in range(32):

returnVect[0,32\*i+j]=int(linestr[j])

return returnVect

### 1.1.4：测试算法

def **handwritingClassTest**():

hwLabels=[]

trainingFieList=listdir(*'trainingDigits'*)

m=len(trainingFieList)

trainingMat=zeros((m,1024))

for i in range(m):

fileNameStr=trainingFieList[i]

fileStr=fileNameStr.split(*'.'*)[0]

classNumStr=int(fileStr.split(*'\_'*)[0])

hwLabels.append(classNumStr)

trainingMat[i,:]=img2vector(*'trainingDigits/%s'*%fileNameStr)

testFileList=listdir(*'testDigits'*)

errorCount=0.0

mTest=len(testFileList)

for i in range(mTest):

fileNameStr=testFileList[i]

fileStr=fileNameStr.split(*'.'*)[0]

classNumStr=int(fileStr.split(*'\_'*)[0])

vectorUnderTest=img2vector(*'testDigits/%s'*%fileNameStr)

classifierResult=classify0(vectorUnderTest,trainingMat, hwLabels, 3)

print(*"the classifier came back with: %d, the real answer is: %d "*%(classifierResult,classNumStr))

if classifierResult!=classNumStr: errorCount+=1

print(*"\n the total number of errors is : %d"*%errorCount)

print(*"\n the total error rate is : %f"*%(errorCount/float(mTest)))

### 1.1.5：运行结果

the classifier came back with: 1, the real answer is: 1

the classifier came back with: 1, the real answer is: 1

the classifier came back with: 1, the real answer is: 1

......

the classifier came back with: 9, the real answer is: 9

the classifier came back with: 9, the real answer is: 9

the classifier came back with: 9, the real answer is: 9

the classifier came back with: 4, the real answer is: 9

**the total number of errors is : 31**

**the total error rate is : 0.032770**

## 2.1：K-NN算法之TensorFlow框架之 线性处理 **准确率: 0.91**

### 2.1.1：TensorFlow手写数字识别大致步骤

1. 将要识别的图片转为灰度图，并且转化为28\*28矩阵（单通道，每个像素范围0-255，0为黑色，255为白色，这一点与MNIST中的正好相反）
2. 将28\*28的矩阵转换成1维矩阵（也就是把第2,3,4,5....行矩阵纷纷接入到第一行的后面）
3. 用一个1\*10的向量代表标签，也就是这个数字到底是几，举个例子数字1对应的矩阵就是[0,1,0,0,0,0,0,0,0,0]
4. softmax回归预测图片是哪个数字的概率
5. 用交叉熵和梯度下降法训练参数

### [2.1.2：](http://yann.lecun.com/exdb/mnist/)**[数据集](http://yann.lecun.com/exdb/mnist/)**准备

train-images-idx3-ubyte.gz: 训练集-图片，6w

train-labels-idx1-ubyte.gz: 训练集-标签，6w

t10k-images-idx3-ubyte.gz: 测试集-图片，1w

t10k-labels-idx1-ubyte.gz: 测试集-标签，1w

### 2.1.3：输入和输出、损失函数

对于识别mnist图片而言。

输入：784（=28X28）向量

输出：10（概率向量，概率最大的位置，就是预测的数字）

损失函数：即评价模型函数，评估网络模型的好坏，值越大，表示模型越差，反之，越好。常见的有**交叉熵**

### 2.1.4：回归模型

我们可以将网络理解为一个函数，回归模型，其实是希望对这个函数进行拟合。 比如定义模型为 Y = X \* w + b，对应的损失

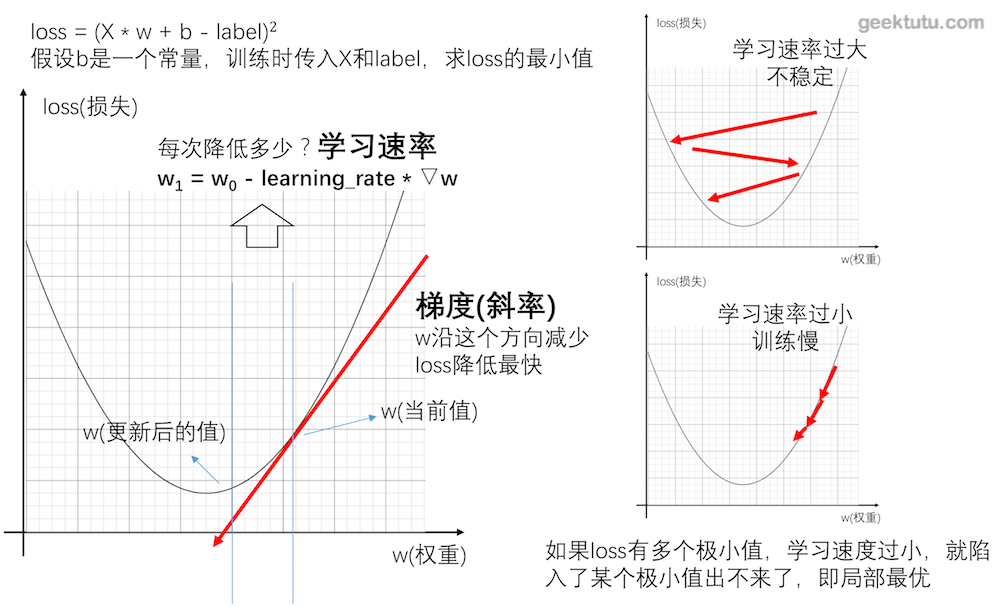
**loss = (Y - labal)^2 = -(X \* w - b - label)^2**

**这里损失函数用方差计算，这个函数是关于w和b的二次函数，所以神经网络训练的目的是找到w和b，使得loss最小。**

**可以通过不断地传入X和label的值，来修正w和b，使得最终得到的Y与label的loss最小。这个训练的过程，可以采用**梯度下降**的方法。通过梯度下降，找到最快的方向，调整w和b值，使得w \* X + b的值越来越接近label。**

### 2.1.5：递归下降、学习率

[详见优化器文档](../优化器Optimizer比较.doc)



**学习速率**

简单说，梯度即一个函数的斜率，找到函数的斜率，其实就知道了w和b的值往哪个方向调整，能够让函数值（loss）降低得最快。那么方向知道了，往这个方向调整多少呢？这个数，神经网络中称之为学习速率。学习速率调得太低，训练速度会很慢，学习速率调得过高，每次迭代波动会很大。

**可以使用Adam（Adaptive Moment Estimation）**

**优化算法比较：**

[详见优化器文档](../优化器Optimizer比较.doc)

### 2.1.6：代码实现

构造网络**model.py**

**import tensorflow as tf**

**class NetWork(object):**

**def \_\_init\_\_(*self*):**

**# 学习速率，一般在 0.00001 - 0.5 之间**

***self*.learning\_rate = 0.001**

**# 输入张量 28 \* 28 = 784个像素的图片一维向量**

***self*.x = tf.placeholder(tf.float32, [None, 784])**

**# 标签值，即图像对应的结果，如果对应数字是8，则对应label是 [0,0,0,0,0,0,0,0,1,0]**

**# 这种方式称为 one-hot编码**

**# 标签是一个长度为10的一维向量，值最大的下标即图片上写的数字**

***self*.label = tf.placeholder(tf.float32, [None, 10])**

**# 权重，初始化全 0**

***self*.w = tf.Variable(tf.zeros([784, 10]))**

**# 偏置 bias， 初始化全 0**

***self*.b = tf.Variable(tf.zeros([10]))**

**# 输出 y = softmax(X \* w + b)**

***self*.y = tf.nn.softmax(tf.matmul(*self*.x, *self*.w) + *self*.b)**

**# 损失，即交叉熵，最常用的计算标签(label)与输出(y)之间差别的方法**

***self*.loss = -tf.reduce\_sum(*self*.label \* tf.log(*self*.y + 1e-10))**

**# 反向传播，采用梯度下降的方法。调整w与b，使得损失(loss)最小**

**# loss越小，那么计算出来的y值与 标签(label)值越接近，准确率越高**

***self*.train = tf.train.GradientDescentOptimizer(*self*.learning\_rate).minimize(*self*.loss)**

**# 以下代码验证正确率时使用**

**# argmax 返回最大值的下标，最大值的下标即答案**

**# 例如 [0,0,0,0.9,0,0.1,0,0,0,0] 代表数字3**

**predict = tf.equal(tf.argmax(*self*.label, 1), tf.argmax(*self*.y, 1))**

**# predict -> [true, true, true, false, false, true]**

**# reduce\_mean即求predict的平均数 即 正确个数 / 总数，即正确率**

***self*.accuracy = tf.reduce\_mean(tf.cast(predict, *"float"*))**

**训练train.py**

#coding:utf-8

from KNN.TensorFlow.model import NetWork

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

class **Train**(object):

def **\_\_init\_\_**(*self*):

*self*.net=NetWork()

# 初始化 session

# Network() 只是构造了一张计算图，计算需要放到会话(session)中

*self*.sess = tf.Session()

# 初始化变量

*self*.sess.run(tf.global\_variables\_initializer())

# 读取训练和测试数据，tensorflow库自带的，不存在训练集会自动下载

# 项目目录下已经下载好，删掉后，重新运行代码会自动下载

*self*.data = input\_data.read\_data\_sets(*'data\_set'*, one\_hot=True)

def **train**(*self*):

# batch\_size 是指每次迭代训练，传入训练的图片张数。

# 数据集小，可以使用全数据集，数据大的情况下，

# 为了提高训练速度，用随机抽取的n张图片来训练，效果与全数据集相近

# https://www.zhihu.com/question/32673260

batch\_size = 64

# 总的训练次数

train\_step = 2000

# 开始训练

for i in range(train\_step):

# 从数据集中获取 输入和标签(也就是答案)

x, label = *self*.data.train.next\_batch(batch\_size)

# 每次计算train，更新整个网络

# loss只是为了看到损失的大小，方便打印

\_, loss = *self*.sess.run([*self*.net.train, *self*.net.loss],

feed\_dict={*self*.net.x: x, *self*.net.label: label})

# 打印 loss，训练过程中将会看到，loss有变小的趋势

# 代表随着训练的进行，网络识别图像的能力提高

# 但是由于网络规模较小，后期没有明显下降，而是有明显波动

if (i + 1) % 10 == 0:

print(*'第%5d步，当前loss：%.2f'* % (i + 1, loss))

def **calculate\_accuracy**(*self*):

test\_x = *self*.data.test.images

test\_label = *self*.data.test.labels

# 注意：与训练不同的是，并没有计算 self.net.train

# 只计算了accuracy这个张量，所以不会更新网络

# 最终准确率约为0.91

accuracy = *self*.sess.run(*self*.net.accuracy,

feed\_dict={*self*.net.x: test\_x, *self*.net.label: test\_label})

print(*"准确率: %.2f，共测试了%d张图片 "* % (accuracy, len(test\_label)))

**测试\_\_init\_\_.py**

#coding:utf-8

from KNN.TensorFlow.train import Train

if \_\_name\_\_==*"\_\_main\_\_"*:

app=Train()

app.train()

app.calculate\_accuracy()

### 2.1.7：运行结果

第 10步，当前loss：117.76

第 20步，当前loss：93.58

第 30步，当前loss：78.34

第 40步，当前loss：68.88

第 50步，当前loss：56.77

第 60步，当前loss：53.12

第 70步，当前loss：49.78

第 80步，当前loss：54.04

......

第 1960步，当前loss：14.80

第 1970步，当前loss：18.13

第 1980步，当前loss：16.06

第 1990步，当前loss：16.24

第 2000步，当前loss：25.67

准确率: 0.91，共测试了10000张图片

## 3.1：K-NN算法之TensorFlow框架之 CNN 处理 **准确率: 1**

### 3.1.1：TensorFlow手写数字识别大致步骤

使用一个简单的CNN网络结构如下，括号里边表示tensor经过本层后的输出shape：

* 输入层（28 \* 28 \* 1）
* 卷积层1（28 \* 28 \* 32）
* pooling层1（14 \* 14 \* 32）
* 卷积层2（14 \* 14 \* 64）
* pooling层2（7 \* 7 \* 64）
* 全连接层（1 \* 1024）
* softmax层（10）

流程如下：

* Layer1：

input[28x28x1]->conv[5x5，(1:32)]->relu[28x28x32]->

max\_pool[2x2，14x14x32]->dropout[0.75]

* Layer2：   
  [14x14x32]->conv[5x5，(32:64)]->relu[14x14x64]->

max\_pool[2x2，7x7x64]->dropout[0.75]

* Layer3：

[7x7x64]->FC[1024]->relu->dropout->y = wx+b[10]

### [3.1.2：](http://yann.lecun.com/exdb/mnist/)**[数据集](http://yann.lecun.com/exdb/mnist/)**准备

train-images-idx3-ubyte.gz: 训练集-图片，6w

train-labels-idx1-ubyte.gz: 训练集-标签，6w

t10k-images-idx3-ubyte.gz: 测试集-图片，1w

t10k-labels-idx1-ubyte.gz: 测试集-标签，1w

### 3.1.3：输入和输出、损失函数

对于识别mnist图片而言。

输入：784（=28X28）向量

输出：10（概率向量，概率最大的位置，就是预测的数字）

损失函数：即评价模型函数，评估网络模型的好坏，值越大，表示模型越差，反之，越好。常见的有**交叉熵**

### 3.1.4：重要函数说明

先对几个会用到的主要函数中的主要参数进行说明

* tf.truncated\_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)

随机产生一个形状为**shape的服从截断正态分布**（均值：mean，标准差：stddev），截断的方法根据官方API的定义为，如果单次随机生成的值偏离均值2倍标准差之外，就丢弃并从新随机生成一个新的数

* tf.nn.conv2d(input, filter, strides, padding, use\_cudnn\_on\_gpu=None, data\_format=None, name=None)
* input

input是一个形状为[batch, in\_height, in\_width, in\_channels]的tensor：

* batch 每次batch数据的数量。
* in\_height，in\_width  
  输入矩阵的高和宽，如输入层的图片是28\*28，则in\_height和in\_width就都为28。
* in\_channels  
  输入通道数量。如输入层的图片经过了二值化，则通道为1，如果输入层的图片是RGB彩色的，则通道为3；再如卷积层1有32个通道，则pooling层1的输入（卷积层1的输出）即为32通道。
* filter

filter是一个形状为[filter\_height, filter\_width, in\_channels, out\_channels]的tensor：

* filter\_height, filter\_width  
  卷积核的高与宽。如卷积层1中的卷积核，filter\_height, filter\_width都为28。
* in\_channels 输入通道数量。
* out\_channels  
  输出通道的数量。如输入数据经过卷积层1后，通道数量从1变为32。
* strides

strides是指滑动窗口（卷积核）的滑动规则，包含4个维度，分别对应input的4个维度，即每次在input tensor上滑动时的步长。其中batch和in\_channels维度一般都设置为1，所以形状为[1, stride, stride, 1]。

* padding

<http://www.cnblogs.com/willnote/p/6746668.html>

* tf.nn.max\_pool(value, ksize, strides, padding, data\_format, name)
* value 以tf.nn.conv2d()函数的参数input理解即可。
* ksize  
   滑动窗口（pool）的大小尺寸，这里注意这个大小尺寸并不仅仅指2维上的高和宽，ksize的每个维度同样对应input的各个维度（只是大小，不是滑动步长），同样的，batch和in\_channels维度多设置为1。如pooling层1的ksize即为[1, 2, 2, 1]，即用一个2\*2的窗口做pooling。
* strides  
  同tf.nn.conv2d()函数的参数strides。
* padding
* tf.nn.dropout(x, keep\_prob, noise\_shape, seed, name)

就是dropout